An Algebraic Theory for Web Service Contracts

An Algebraic Theory for Web Service Contracts

Cosimo Laneve[1] and Luca Padovani[2]

[1] Università di Bologna – INRIA Focus Team, Italy
[2] Università di Torino – Dipartimento di Informatica, Italy

Abstract. We study a natural notion of compliance between clients and services in terms of their BPEL (abstract) descriptions. The induced preorder shows interesting connections with the *must* preorder and has normal form representatives that are parallel-free finite-state activities, called *contracts*. The preorder also admits the notion of least service contract that is compliant with a client contract, called *principal dual contract*. Our framework serves as a foundation of Web service technologies for connecting abstract and concrete service definitions and for service discovery.

1 Introduction

Service-oriented technologies and Web services have been proposed as a new way of distributing and organizing complex applications across the Internet. These technologies are nowadays extensively used for delivering cloud computing platforms.

A large effort in the development of Web services has been devoted to their specification, their publication, and their use. In this context, the Business Process Execution Language for Web Services (BPEL for short) has emerged as the de-facto standard for implementing Web service composition and, for this reason, it is supported by the toolkits of the main software vendors (Oracle Process Manager, IBM WebSphere, and Microsoft BizTalk).

As regards publication, service descriptions should retain abstract (behavioral) definitions, which are separate from the binding to a concrete protocol or endpoint. The current standard is defined by the Web Service Description Language (WSDL) [10], which specifies the format of the exchanged messages – the *schema* –, the locations where the interactions are going to occur – the *interface* –, the transfer mechanism to be used (i.e. SOAP-RPC, or others), and basic service abstractions (one-way/asynchronous and request-response/synchronous patterns of conversations). Since these abstractions are too simple for expressing arbitrary, possibly cyclic protocols of exchanged messages between communicating parties, WSDL is not adequate to verify the behavioral compliance between parties. It is also worth to notice that the attempts, such as UDDI (Universal Description, Discovery and Integration) registries [5], provide limited support because registry items only include pointers to the locations of the service abstractions, without constraining the way these abstractions are defined or related to the actual implementations (*cf.* the `<tModel>` element). In this respect, UDDI

registries are almost useless for discovering services; an operation that is performed manually by service users and consumers.

The publication of abstract service descriptions, called *contracts* in the following, and the related ability of service searching assume the existence of a formal notion of contract equivalence and, more generally, of a formal theory for reasoning about Web services by means of their contracts. We identify three main goals of a theory of Web service contracts: (1) it should provide a formal language for describing Web services at a reasonable level of abstraction and for admitting static correctness verification of client/service protocol implementations; (2) it should provide a semantic notion of contract equivalence embodying the principle of safe Web service replacement. Indeed, the lack of a formal characterization of contracts only permits excessively demanding notions of equivalence such as nominal or structural equality; (3) it should provide tools for effectively and efficiently searching Web services in Web service repositories according to their contract.

The aim of this contribution is to provide a suitable theory of contracts for Web services by developing a semantic notion of contract equivalence. In fact, we relax the equivalence into a *subcontract preorder*, so that Web services exposing "larger" contracts can be *safely* returned as results of queries for Web services with "smaller" contracts. We will precisely define what "smaller" and "larger" mean, and we will define which safety property we wish to preserve when substituting a service exposing a contract with a service exposing a larger contract. Our investigation abstracts away from the syntactical details of schemas as well as from those aspects that are oriented to the actual implementations, such as the definition of transmission protocols; all these aspects may be easily integrated on top of the formalism. We do not commit to a particular interpretation of the actions occurring in contracts either: they can represent different typed channels on which interaction occurs or different types of messages.

To equip contracts with a subcontract preorder, we commit to a testing approach. We define client satisfaction as the ability of the client to successfully complete *every* interaction with the service; here "successfully" means that the client never gets stuck (this notion is purposefully asymmetric as client's satisfaction is our main concern). The preorder arises by comparing the sets of clients satisfied by services.

The properties enjoyed by the subcontract preorder are particularly relevant in the context of Web services. Specifically, it is possible to determine, given a client exposing a certain behavior, the smallest (according to subcontract preoder) service contract that satisfies the client – the *principal dual contract*. This contract, acting like a *principal type* in type systems, guarantees that a query to a Web service registry is answered with the largest possible set of compatible services in the registry's databases.

Related works. Our contracts are normal forms of τ-less CCS processes, a calculus developed by De Nicola and Hennessy in a number of contributions [13, 15, 18]. The use of formal models to describe communication protocols is not new (see for instance the exchange patterns in SSDL [20], which are based on CSP and the π-

calculus), nor is it the use or CCS processes as behavioral types (see [19] and [9]). The subcontract relation $\lesssim$ has been introduced in [17]. In [6] the authors have studied a refined version of $\lesssim$ that is more suited for orchestrations. The works that are more closely related to ours are by Castagna *et al.* [8] and the ones on *session types*, especially [14] by Gay and Hole. The authors of [8] make the assumption that client and service can be mediated by a *filter*, which prevents potentially dangerous interactions by dynamically changing the interface of the service as it is seen by the client. The present work can be seen as a special case of [8] in which the filter is static and consequently is unnecessary; at the same time, in the present work we also consider divergence, which is not addressed in [8]. With respect to [14] (and systems based on session types) our contract language is much simpler and it can express more general forms of interaction. While the language defined in [14] supports first-class sessions and name passing, it is purposefully tailored so that the transitivity problems mentioned above are directly avoided at the language level. This restricts the subcontract relation in such a way that internal and external choices can never be related (hence, $\{a, b\} : a \oplus b \preceq \{a, b\} : a + b$ does *not* hold).

As regards schemas, which are currently part of BPEL contracts, it is worth mentioning that they have been the subject of formal investigation by several research projects [4, 16]. This work aims at pursuing a similar objective, but moving from the description of data to the description of behaviors.

Structure of the paper. In Section 2 we introduce BPEL abstract activities and their semantics. In Section 3 we define contracts and detail their relationship with BPEL abstract activities. In Section 4 we address the issue of service discovery in repositories. We conclude in Section 5. Due to space limitations, proofs have been omitted; they can be found in the full paper.

2 BPEL and the abstract language

We introduce the basic notions of BPEL by means of an example. The XML document in Figure 1 describes the behavior of an order service that interacts with four other partners, one of them being the customer (`purchasing`), the others being providers of price (`invoicing` service), shipment (`shipping` service), and manufacturing scheduling (`scheduling` service). The business process is made of *activities*, which can be either *atomic* or *composite*. In this example atomic activities are represented by *invocation* of operations in other partners (lines 7–9, 15–18, 22–25), *acceptance* of messages from other partners, either as incoming requests (line 3) or as responses to previous invocations (lines 10–12 and 19), and *sending* of responses to clients (line 28). Atomic activities are composed together into so-called structured activities, such as *sequential composition* (see the `sequence` fragments) and *parallel composition* (see the `flow` fragment at lines 4–27). In a `sequence` fragment, all the child activities are executed in the order in which they appear, and each activity begins the execution only after the previous one has completed. In a `flow` fragment, all the child activities are executed in

```
1  <process>
2   <sequence>
3     <receive partnerLink="purchasing" operation="sendPurchaseOrder"/>
4     <flow>
5       <links> <link name="ship-to-invoice"/> <link name="ship-to-scheduling"/> </links>
6       <sequence>
7         <invoke partnerLink="shipping" operation="requestShipping">
8           <sources> <source linkName="ship-to-invoice"/> </sources>
9         </invoke>
10        <receive partnerLink="shipping" operation="sendSchedule">
11          <sources> <source linkName="ship-to-scheduling"/> </sources>
12        </receive>
13      </sequence>
14      <sequence>
15        <invoke partnerLink="invoicing" operation="initiatePriceCalculation"/>
16        <invoke partnerLink="invoicing" operation="sendShippingPrice">
17          <targets> <target linkName="ship-to-invoice"/> </targets>
18        </invoke>
19        <receive partnerLink="invoicing" operation="sendInvoice"/>
20      </sequence>
21      <sequence>
22        <invoke partnerLink="scheduling" operation="requestProductionScheduling"/>
23        <invoke partnerLink="scheduling" operation="sendShippingSchedule">
24          <targets> <target linkName="ship-to-scheduling"/> </targets>
25        </invoke>
26      </sequence>
27    </flow>
28    <reply partnerLink="purchasing" operation="sendPurchaseOrder"/>
29  </sequence>
30 </process>
```

Fig. 1. BPEL business process for an e-commerce service.

parallel, and the whole `flow` activity completes as soon as *all* the child activities have completed. It is possible to constrain the execution of parallel activities by means of *links*. In the example, there is a link `ship-to-invoice` declared at line 5 and used in lines 8 and 17, meaning that the invocation at lines 16–18 cannot take place before the one at lines 7–9. Similarly, the link `ship-to-scheduling` means that the invocation at lines 23–25 cannot take place before the receive operation at lines 10–12 has completed. Intuitively, the presence of links limits the possible interleaving of the activities in a `flow` fragment.

Note that BPEL includes other conventional constructs not shown in the example, such as conditional and iterative execution of activities.

To pursue our formal investigation, we will now present an abstract language of processes whose operators correspond to those found in BPEL. Since we will focus on the interactions of BPEL activities with the external environment, rather than on the actual implementation of business processes, our process language overlooks details regarding internal, unobservable evaluations. For example, the BPEL activity

```
<if>
  <condition> bool-expr </condition>
  activity-True
  <else> activity-False </else>
</if>
```

will be abstracted into the process `activity-True` $\oplus$ `activity-False`, meaning that one of the two activities will be performed and the choice will be a conse-

quence of some unspecified internal decision. A similar observation pertains to the `<while>` activity (see Remark 2.2).

2.1 Syntax of BPEL abstract activities

Let N be a set of *names*, ranged over by $a, b, c, \ldots$, and $\overline{\mathsf{N}}$ be a disjoint set of *co-names*, ranged over by $\overline{a}, \overline{b}, \overline{c}, \ldots$; the term *action* refers to names and co-names without distinction; actions are ranged over by $\alpha, \beta, \ldots$. Let $\overline{\overline{a}} = a$. We use $\varphi, \psi, \ldots$ to range over $(\mathsf{N} \cup \overline{\mathsf{N}})^*$ and $\mathrm{R}, \mathrm{S}, \ldots$ to range over finite sets of actions. Let $\overline{\mathrm{R}} \stackrel{\text{def}}{=} \{\overline{\alpha} \mid \alpha \in \mathrm{R}\}$. The syntax of BPEL *abstract activities* is defined by the following grammar:

$$
\begin{array}{rcll}
P, Q, P_i & ::= & \mathsf{0} & (\texttt{empty}) \\
& | & a & (\texttt{receive}) \\
& | & \overline{a} & (\texttt{invoke}) \\
& | & \sum_{i \in I} \alpha_i \,;\, P_i & (\texttt{pick}) \\
& | & P \mid_A Q & (\texttt{flow \& link}) \\
& | & P \,;\, Q & (\texttt{sequence}) \\
& | & \bigoplus_{i \in I} P_i & (\texttt{if}) \\
& | & P^{\boldsymbol{*}} & (\texttt{while})
\end{array}
$$

Each construct is called with the name of the corresponding BPEL construct. The activity $\mathsf{0}$ represents the completed process, it performs no visible action. The activity a represents the act of waiting for an incoming message. Here we take the point of view that a stands for a particular operation implemented by the process. The activity $\overline{a}$ represents the act of invoking the operation a provided by another partner. The activity $\sum_{i \in I} \alpha_i \,;\, P_i$ represents the act of waiting for any of the α_i operations to be performed, i belonging to a *finite* set I. Whichever operation α_i is performed, it first disables the remaining ones and the continuation P_i is executed. If $\alpha_i = \alpha_j$ and $i \neq j$, then the choice whether executing P_i or P_j is implementation dependent. The process $P \mid_A Q$, where A is a set of names, represents the parallel composition (`flow`) of P and Q and the creation of a private set A of `link` names that will be used by P and Q to synchronize; an example will be given shortly. The n-ary version $\prod^A_{i \in 1..n} P_i$ of this construct may also be considered: we stick to the binary one for simplicity. The process $P \,;\, Q$ represents the sequential composition of P followed by Q. Again we only provide a binary operator, where the BPEL one is n-ary. The process $\bigoplus_{i \in I} P_i$, again with I finite, represents an internal choice performed by the process, that results into one of the I exclusive continuations P_i. Finally, $P^{\boldsymbol{*}}$ represents the repetitive execution of process P so long as an internally verified condition is satisfied.

The pick activity $\sum_{i \in 1..n} \alpha_i \,;\, P_i$ and the if activity $\bigoplus_{i \in 1..n} P_i$ will be also written $\alpha_1 \,;\, P_1 + \cdots + \alpha_n \,;\, P_n$ and $P_1 \oplus \cdots \oplus P_n$, respectively. In the following we treat (`empty`), (`receive`), and (`invoke`) as special cases of (`pick`), while at the same time keeping the formal semantics just as easy. In particular, we write $\mathsf{0}$ for $\sum_{\alpha \in \varnothing} \alpha \,;\, P_\alpha$ and α as an abbreviation for $\sum_{\alpha \in \{\alpha\}} \alpha \,;\, \mathsf{0}$ (tailing $\mathsf{0}$ are always omitted). Let also $\mathsf{actions}(P)$ be the set of actions occurring in P.

Table 1. Legend for the operations of the BPEL process in Figure 1.

Name	Operation
a	`sendPurchaseOrder`
b	`requestShipping`
c	`sendSchedule`
d	`initiatePriceCalculation`
e	`sendShippingPrice`
f	`sendInvoice`
g	`requestProductionScheduling`
h	`sendShippingSchedule`
x	`ship-to-invoce`
y	`ship-to-scheduling`

Example 2.1. Table 1 gives short names to the operations used in the business process shown in Figure 1. Then the whole BPEL activity can be described by the term

$$a\,;\left(\overline{b}\,;\left(\left(\overline{x}\mid_{\varnothing} c\,;\overline{y}\right)\mid_{\{x\}}\overline{d}\,;x\,;\overline{e};f\right)\mid_{\{y\}}\overline{g}\,;y\,;\overline{h}\right);\overline{a}$$

where we use names for specifying links. Such names are restricted so that they are not visible from outside. Indeed, they are completely internal to the process and should not be visible in the process' contract. ◊

Remark 2.1. The BPEL specification defines a number of static analysis requirements beyond the mere syntactic correctness of processes whose purpose is to "detect any undefined semantics or invalid semantics within a process definition" [2]. Several of these requirements regard the use of links. For example, it is required that no link must cross the boundary of a repeatable construct (`while`). It is also required that link ends must be used exactly once (hence $\mathsf{0}\mid_{\{a\}} a$ is invalid because $\overline{a}$ is never used), and the dependency graph determined by links must be acyclic (hence $a.\overline{b}\mid_{\{a,b\}} b.\overline{a}$ is invalid because it contains cycles). These constraints may be implemented by restricting the arguments to the above abstract activities and then using static analysis techniques. ■

2.2 Operational semantics of BPEL abstract activities

The operational semantics of BPEL abstract activities is defined by means of a completion predicate and of a labelled transition system. Let $P\checkmark$, read *P has completed*, be the least predicate such that

$$\mathsf{0}\checkmark \qquad \frac{P\checkmark \quad Q\checkmark}{P\mid_A Q\checkmark} \qquad \frac{P\checkmark \quad Q\checkmark}{P\,;Q\checkmark}$$

Let μ range over actions and the special name ε denote internal moves. The operational semantics of processes is described by the following rules plus the

symmetric of the rules for $|$.

$$\frac{}{\sum_{i\in I}\alpha_i\,;P_i \xrightarrow{\alpha_i} P_i}\ (\textsc{action}) \qquad \frac{}{\bigoplus_{i\in I} P_i \xrightarrow{\varepsilon} P_i}\ (\textsc{if})$$

$$\frac{P \xrightarrow{\mu} P' \quad \mu \notin \mathrm{A}\cup\overline{\mathrm{A}}}{P\mid_A Q \xrightarrow{\mu} P'\mid_A Q}\ (\textsc{flow}) \qquad \frac{P \xrightarrow{\alpha} P' \quad Q \xrightarrow{\overline{\alpha}} Q' \quad \alpha \in \mathrm{A}\cup\overline{\mathrm{A}}}{P\mid_A Q \xrightarrow{\varepsilon} P'\mid_A Q'}\ (\textsc{link})$$

$$\frac{P \xrightarrow{\mu} P'}{P\,;Q \xrightarrow{\mu} P'\,;Q}\ (\textsc{seq}) \qquad \frac{P\checkmark \quad Q \xrightarrow{\mu} Q'}{P\,;Q \xrightarrow{\mu} Q'}\ (\textsc{seq-end}) \qquad \frac{}{P^{*} \xrightarrow{\varepsilon} \mathsf{0}}\ (\textsc{while-end}) \qquad \frac{P \xrightarrow{\mu} P'}{P^{*} \xrightarrow{\mu} P'\,;P^{*}}\ (\textsc{while})$$

We briefly describe the rules. The process $\sum_{i\in I}\alpha_i\,;P_i$ has as many α-labelled transitions as the number of actions in $\{\alpha_i \mid i\in I\}$. After a visible transition, only the selected continuation is allowed to execute. The process $\bigoplus_{i\in I} P_i$ may internally choose to behave as one of the P_i, with $i\in I$. The process $P\mid_A Q$ allows P and Q to internally evolve autonomously, or to emit messages, or to synchronize with each other on names in the set A. It completes when both P and Q have completed. The process $P\,;Q$ reduces according to the reductions of P first, and of Q when P has completed. Finally, the process P^{*} may either complete in one step by reducing to $\mathsf{0}$, or it may execute P one more time followed by P^{*}. The choice among the two possibilities is performed internally.

Remark 2.2. According to the operational semantics, P^{*} may execute the activity P an arbitrary number of times. This is at odds with concrete BPEL activities having P^{*} as abstract counterpart. For example, the BPEL activity

```
<while>
  <condition> bool-expr </condition>
    activity
</while>
```

executes `activity` as long as the `bool-expr` condition is true. Representing such BPEL activity with $\texttt{activity}^{*}$ means *overapproximating* it. This abstraction is crucial for the decidability of our theory. ■

We illustrate the semantics of BPEL abstract activities through a couple of examples:

1. $(\overline{a}\oplus\overline{b}\mid_{\{a,b\}} a\oplus b)\,;\overline{c} \xrightarrow{\varepsilon} (\overline{a}\mid_{\{a,b\}} a\oplus b)\,;\overline{c}$ by (IF), (FLOW), and (SEQ). By the same rules, it is possible to have $(\overline{a}\mid_{\{a,b\}} a\oplus b)\,;\overline{c} \xrightarrow{\varepsilon} (\overline{a}\mid_{\{a,b\}} b)\,;\overline{c}$, which cannot reduce anymore ($\overline{a}\mid_{\{a,b\}} b$ is a *deadlocked* activity).
2. let $\Psi \stackrel{\text{def}}{=} \mathsf{0}\,;(\mathsf{0}\oplus\mathsf{0})^{*}$. Then, according to rules (SEQ-END), (IF), and (WHILE), $\Psi \xrightarrow{\varepsilon} \Psi$ and $\Psi \xrightarrow{\varepsilon} \mathsf{0}$.

Let $\stackrel{\varepsilon}{\Longrightarrow}$ be the reflexive, transitive closure of $\xrightarrow{\varepsilon}$ and $\stackrel{\alpha}{\Longrightarrow}$ be $\stackrel{\varepsilon}{\Longrightarrow}\xrightarrow{\alpha}\stackrel{\varepsilon}{\Longrightarrow}$; let also $P \xrightarrow{\mu}$ (resp. $P \stackrel{\alpha}{\Longrightarrow}$) if there exists P' such that $P \xrightarrow{\mu} P'$ (resp. $P \stackrel{\alpha}{\Longrightarrow} P'$); we let $P \not\xrightarrow{\mu}$ if not $P \xrightarrow{\mu}$.

A relevant property of our BPEL abstract calculus is that the model of every activity is always finite. This result is folklore (the argument is similar to the one for CCS$_*$ [7]).

Lemma 2.1. *Let* $Reach(P) = \{Q \mid \text{there are } \mu_1, \ldots, \mu_n \text{ with } P \xrightarrow{\mu_1} \cdots \xrightarrow{\mu_n} Q\}$. *Then, for every activity* P, *the set* $Reach(P)$ *is always finite.*

We introduce a number of auxiliary definitions that will be useful in Section 3. By Lemma 2.1 these notions are trivially decidable.

Definition 2.1. *We introduce the following notation:*

- $P{\uparrow}$ *if there is an infinite sequence of* ε*-transitions* $P \xrightarrow{\varepsilon} \xrightarrow{\varepsilon} \cdots$ *starting from* P. *Let* $P{\downarrow}$ *if not* $P{\uparrow}$.
- $\mathsf{init}(P) \stackrel{\text{def}}{=} \{\alpha \mid P \stackrel{\alpha}{\Longrightarrow}\}$;
- *we say that* P *has* ready set R, *notation* $P \Downarrow \mathrm{R}$, *if* $P \stackrel{\varepsilon}{\Longrightarrow} P'$ *and* $\mathrm{R} = \mathsf{init}(P')$;
- *let* $P \stackrel{\alpha}{\Longrightarrow}$. *Then* $P(\alpha) \stackrel{\text{def}}{=} \bigoplus_{P \stackrel{\varepsilon}{\Longrightarrow} \xrightarrow{\alpha} P'} P'$. *We call* $P(\alpha)$ *the* continuation of P after α.

The above definitions are almost standard, except for $P(\alpha)$ (that we already used in [17]). Intuitively, $P(\alpha)$ represents the residual behavior of P after an action α, from the point of view of the party that is interacting with P. Indeed, the party does not know which, of the possibly multiple, α-labelled branches P has taken. For example $(a\,;b{+}a\,;c{+}b\,;d)(a) = b \oplus c$ and $(a\,;b{+}a\,;c{+}b\,;d)(b) = d$.

2.3 The compliance preorder

We proceed defining a notion of equivalence between activities that is based on their observable behavior. To this aim, we introduce a special name e for denoting the successful termination of an activity ("e" stands for `end`). By *compliance* between a "client" activity T and a "service" activity P we mean that every interaction between T and P where P stops communicating with T is such that T has reached a successfully terminated state.

Definition 2.2 (Compliance). *Let* $A_P = \{a \mid a \in \mathsf{actions}(P) \cup \overline{\mathsf{actions}(P)}\}$ *and* $\mathsf{e} \notin A_P$. *The (client) activity* T *is* compliant with *the (service) activity* P, *written* $T \dashv P$, *if* $P \mid_{A_P} T \stackrel{\varepsilon}{\Longrightarrow} P' \mid_{A_P} T'$ *implies:*

1. *if* $P' \mid_{A_P} T' \stackrel{\varepsilon}{\not\longrightarrow}$, *then* $\{\mathsf{e}\} \subseteq \mathsf{init}(T')$, *and*
2. *if* $P'{\uparrow}$, *then* $\{\mathsf{e}\} = \mathsf{init}(T')$.

We write $P \sqsubseteq_{\sim} Q$, *called* compliance preorder, *if and only if* $T \dashv P$ *implies* $T \dashv Q$ *for every* T. *Let* $\asymp \stackrel{\text{def}}{=} \sqsubseteq_{\sim} \cap \sqsupseteq_{\sim}$.

According to the notion of compliance, if the client-service conversation terminates, then the client is in a successful state (it will emit an e-name). For example, $a\,;\mathsf{e}{+}b\,;\mathsf{e} \dashv \overline{a} \oplus \overline{b}$ and $a\,;\mathsf{e}\oplus b\,;\mathsf{e} \dashv \overline{a}{+}\overline{b}$ but $a\,;\mathsf{e}\oplus b\,;\mathsf{e} \not\dashv \overline{a}\oplus\overline{b}$ because of the computation $a\,;\mathsf{e}\oplus b\,;\mathsf{e} \mid_{\{a,b\}} \overline{a}\oplus\overline{b} \Longrightarrow a\,;\mathsf{e} \mid_{\{a,b\}} \overline{b} \not\longrightarrow$ where the client

waits for an interaction on a in vain. Similarly, the client must reach a successful state if the conversation does not terminate but the divergence is due to the service. In this case, however, every reachable state of the client must be such that the only possible action is e. The practical justification of such a notion of compliance derives from the fact that connection-oriented communication protocols (like those used for interaction with Web services) typically provide for an explicit end-of-connection signal. Consider for example the client behavior $\mathsf{e} + \overline{a}\,;\mathsf{e}$. Intuitively this client tries to send a request on the name a, but it can also succeed if the service rejects the request. So $\mathsf{e} + \overline{a}\,;\mathsf{e} \dashv \mathsf{0}$ because the client can detect the fact that the service is not ready to interact on a. The same client interacting with a diverging service would have no way to distinguish a service that is taking a long time to accept the request from a service that is perpetually performing internal computations, hence $\mathsf{e} + \overline{a}\,;\mathsf{e} \not\dashv \Psi$. As a matter of facts, the above notion of compliance makes Ψ the "smallest service" – the one a client can make the least number of assumptions on (this property will be fundamental in the definition of principal dual contract in Section 4). That is $\Psi \sqsubseteq_{\approx} P$, for every P. As another example, we notice that $a\,;b + a\,;c \sqsubseteq_{\approx} a\,;(b \oplus c)$ since, after interacting on a, a client of the smaller service is not aware of which state the service is in (it can be either b or c). Had we picked only one a-derivative of the smaller contract behavior, we would have failed to relate it with the a-derivative of the larger contract, since both $b \not\sqsubseteq_{\approx} b \oplus c$ and $c \not\sqsubseteq_{\approx} b \oplus c$.

As by Definition 2.2, it is difficult to understand the general properties of the compliance preorder because of the universal quantification over all (client) activities T. For this reason, it is convenient to provide an alternative characterization of $\sqsubseteq_{\approx}$ which turns out to be the following:

Definition 2.3. *A* coinductive compliance *is a relation $\mathcal{R}$ such that $P\ \mathcal{R}\ Q$ and $P{\downarrow}$ implies*

1. *$Q{\downarrow}$, and*
2. *$Q \Downarrow \mathrm{R}$ implies $P \Downarrow \mathrm{S}$ for some $\mathrm{S} \subseteq \mathrm{R}$, and*
3. *$Q \stackrel{\alpha}{\Longrightarrow}$ implies $P \stackrel{\alpha}{\Longrightarrow}$ and $P(\alpha)\ \mathcal{R}\ Q(\alpha)$.*

Let $\preceq$ be the largest coinductive compliance relation.

The pre-order $\preceq$ corresponds to the *must-testing preorder* [15] and is also an alternative definition of $\sqsubseteq_{\approx}$:

Theorem 2.1. *$P \sqsubseteq_{\approx} Q$ if and only if $P \preceq Q$.*

3 Contracts

In this section we discuss how to associate a behavioral description, called *contract*, to a BPEL abstract activity. The ultimate goal is being able to reason about properties of BPEL activities by means of the respective contracts.

Contracts use a set of contract names, ranged over $\mathsf{C}, \mathsf{C}', \mathsf{C}_1, \ldots$. A contract is a tuple

$$(\mathsf{C}_1 = \sigma_1, \ldots, \mathsf{C}_n = \sigma_n,\ \sigma)$$

where $\mathsf{C}_j = \sigma_j$ are contract name definitions, σ is the main term, and we assume that there is no chain of definitions of the form $\mathsf{C}_{n_1} = \mathsf{C}_{n_2}$, $\mathsf{C}_{n_2} = \mathsf{C}_{n_3}$, $\ldots$, $\mathsf{C}_{n_k} = \mathsf{C}_{n_1}$. The syntax of σ_j and σ is given by

$$\sigma \quad ::= \quad \mathsf{C} \ \mid\ \alpha\,;\sigma \ \mid\ \sigma + \sigma \ \mid\ \sigma \oplus \sigma$$

where $\mathsf{C} \in \{\mathsf{C}_1, \ldots, \mathsf{C}_n\}$. The contract $\alpha\,;\sigma$ represents sequential composition in the restricted form of prefixing. The operations $+$ and $\oplus$ correspond to `pick` and `if` of BPEL activities, respectively. These operations are assumed to be associative and commutative; therefore we will write $\sigma_1 + \cdots + \sigma_n$ and $\sigma_1 \oplus \cdots \oplus \sigma_n$ without confusion and will sometimes shorten these contracts as $\sum_{i \in 1..n} \sigma_i$ and $\bigoplus_{i \in 1..n} \sigma_i$, respectively. The contract name C is used to model recursive behaviors such as $\mathsf{C} = a\,;\mathsf{C}$. In what follows we will leave contract name definitions implicit and identify a contract $(\mathsf{C}_1 = \sigma_1, \ldots, \mathsf{C}_n = \sigma_n,\ \sigma)$ with its main body σ. We will write $\mathsf{cnames}(\sigma)$ for the set $\{\mathsf{C}_1, \ldots, \mathsf{C}_n\}$ and use the following abbreviations:

- $\mathsf{0} \stackrel{\text{def}}{=} \mathsf{C}_0$, where $\mathsf{C}_0 = \mathsf{C}_0 + \mathsf{C}_0$ represents a terminated activity;
- $\Omega \stackrel{\text{def}}{=} \mathsf{C}_\Omega$, where $\mathsf{C}_\Omega = \mathsf{C}_\Omega \oplus \mathsf{C}_\Omega$ represents divergence, that is a non-terminating activity.

Note that, even if apparently simpler, the contract language *is not* a sublanguage of BPEL abstract activities. For example, Ω cannot be written as a term in the syntax of Section 2.1. Nevertheless, in the following, we demonstrate that contracts provide alternative descriptions (with respect to the preorder $\sqsubseteq_\approx$) to BPEL abstract activities.

The operational semantics of contracts is defined by the rules below:

$$\alpha\,;\sigma \xrightarrow{\alpha} \sigma \qquad \sigma \oplus \rho \xrightarrow{\varepsilon} \sigma$$

$$\frac{\sigma \xrightarrow{\varepsilon} \sigma'}{\sigma + \rho \xrightarrow{\varepsilon} \sigma' + \rho} \qquad \frac{\sigma \xrightarrow{\alpha} \sigma'}{\sigma + \rho \xrightarrow{\alpha} \sigma'} \qquad \frac{\mathsf{C} = \sigma \quad \sigma \xrightarrow{\mu} \sigma'}{\mathsf{C} \xrightarrow{\mu} \sigma'}$$

plus the symmetric of rules $+$ and $\oplus$. Note that $+$ evaluates the branches as long as they can perform invisible actions. This rule is absent in BPEL abstract activities because, there, the branches are always guarded by an action.

Example 3.1. The Web service conversation language WSCL [3] describes *conversations* between two parties by means of an activity diagram (Figure 2). The diagram is made of *interactions* connected with each other by *transitions*. An interaction is a basic one-way or two-way communication between the client and the server. Two-way communications are just a shorthand for two sequential one-way interactions. Each interaction has a *name* and a list of *document types* that can be exchanged during its execution. A transition connects a *source* interaction

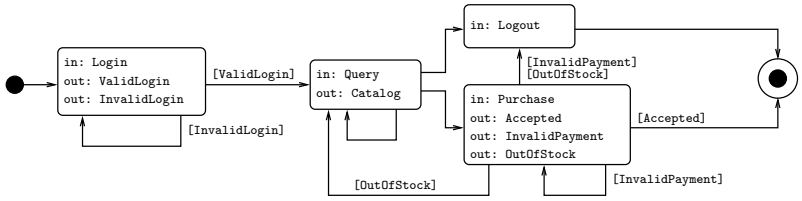

Fig. 2. Contract of a simple e-commerce service as a WSCL diagram.

with a *destination* interaction. A transition may be *labeled* by a document type if it is active only when a message of that specific document type was exchanged during the previous interaction.

The diagram in Figure 2 describes the conversation of a service requiring clients to login before they can issue a query. After the query, the service returns a catalog. From this point on, the client can decide whether to purchase an item from the catalog or to logout and leave. In case of purchase, the service may either report that the purchase is successful, or that the item is out-of-stock, or that the client's payment is refused. By interpreting names as message types, this e-commerce service can be described by the tuple:

$$
\begin{array}{ll}
(& \mathsf{C}_1 = \mathtt{Login}\,;(\overline{\mathtt{InvalidLogin}}\,;\mathsf{C}_1 \oplus \overline{\mathtt{ValidLogin}}\,;\mathsf{C}_2)\ , \\
& \mathsf{C}_2 = \mathtt{Query}\,;\overline{\mathtt{Catalog}}\,;(\mathsf{C}_2+\mathsf{C}_3+\mathsf{C}_4)\ , \\
& \mathsf{C}_3 = \mathtt{Purchase}\,;(\,\overline{\mathtt{Accepted}} \\
& \qquad\qquad \oplus \overline{\mathtt{InvalidPayment}}\,;(\mathsf{C}_3+\mathsf{C}_4) \\
& \qquad\qquad \oplus \overline{\mathtt{OutOfStock}}\,;(\mathsf{C}_2+\mathsf{C}_4)\,)\ , \\
& \mathsf{C}_4 = \mathtt{Logout}\ , \\
& \mathsf{C}_1\ )
\end{array}
$$

There is a strict correspondence between unlabeled (respectively, labeled) transitions in Figure 2 and external (respectively, internal) choices in the contract. Recursion is used for expressing iteration (the cycles in the figure) so that the client is given another chance whenever an action fails for some reason. $\Diamond$

We can relate BPEL abstract activities and contracts by means of the corresponding transition systems. To this aim, let X and Y range over BPEL abstract activities *and* contracts. Then, X and Y interact according to the rules

$$
\frac{\mathsf{X} \xrightarrow{\varepsilon} \mathsf{X}'}{\mathsf{X} \parallel \mathsf{Y} \xrightarrow{\varepsilon} \mathsf{X}' \parallel \mathsf{Y}} \qquad
\frac{\mathsf{Y} \xrightarrow{\varepsilon} \mathsf{Y}'}{\mathsf{X} \parallel \mathsf{Y} \xrightarrow{\varepsilon} \mathsf{X} \parallel \mathsf{Y}'} \qquad
\frac{\mathsf{X} \xrightarrow{\alpha} \mathsf{X}' \quad \mathsf{Y} \xrightarrow{\overline{\alpha}} \mathsf{Y}'}{\mathsf{X} \parallel \mathsf{Y} \xrightarrow{\varepsilon} \mathsf{X}' \parallel \mathsf{Y}'}
$$

It is possible to extend the definition of compliance to contracts and, by Definition 2.2, obtain a relation that allows us to compare activities and contracts without distinction, and similarly for $\preceq$. To be precise, the relation $\mathsf{X} \sqsubseteq_{\approx} \mathsf{Y}$ is smaller (in principle) than the relation $\sqsubseteq_{\approx}$ given in Definition 2.2 because, as we have said, the contract language is not a sublanguage of that of activities and, therefore, the set of tests that can be used for comparing X and Y is larger. Nonetheless, the process language used in [12] includes both BPEL abstract activities and contracts and since $\sqsubseteq_{\approx}$ is equivalent to must-testing, then we may

safely use the same symbol $\sqsubseteq$ for both languages. This is a key point in our argument, which will allow us to define, for every activity P, a contract σ_P such that $P \asymp \sigma_P$. In particular, let C_P be the set of contract name definitions defined as follows

$$\mathsf{C}_P = \begin{cases} \Omega & \text{if } P\!\uparrow \\ \bigoplus_{P \Downarrow \mathrm{R}} \sum_{\alpha \in \mathrm{R}} \alpha \,;\, \mathsf{C}_{P(\alpha)} & \text{otherwise} \end{cases}$$

A relevant property of C_P is an immediate consequence of Lemma 2.1.

Lemma 3.1. *For every P, the set $\mathsf{cnames}(\mathsf{C}_P)$ is finite.*

The construction of the contract C_P with respect to a BPEL abstract activity is both correct and complete with respect to compliance:

Theorem 3.1. $P \asymp \mathsf{C}_P$.

4 Service discovery and dual contracts

We now turn our attention to the problem of querying a database of Web service contracts. To this aim, the relation $\sqsubseteq$ (and the must-testing) turns out to be too strong (see below). Following [17], we switch to more informative service contracts than what described in Section 3. In particular, we consider pairs $\mathrm{I} : \sigma$, where I is the interface, i.e. the set of actions performed by the service, and σ is as in Section 3 (it is intended that the names occurring in σ are included into I). It is reasonable to think that a similar extension applies to client contracts: clients, which are defined by BPEL activities as well, are abstracted by terms in the language of Section 2 and, in turn, their behavior is defined by a term in the contract language, plus the interface.

Definition 4.1 (Subcontract relation). *Let $\mathrm{I} : \sigma \lesssim \mathrm{J} : \tau$ if $\mathrm{I} \subseteq \mathrm{J}$ and, for every ρ such that $\mathsf{actions}(\rho) \setminus \{\mathsf{e}\} \subseteq \bar{\mathrm{I}}$ and $\rho \dashv \sigma$ implies $\rho \dashv \tau$. Let $\approx$ be $\lesssim \cap \gtrsim$.*

Let us comment on the differences between $\mathrm{I} : \sigma \lesssim \mathrm{J} : \tau$ and $\sigma \sqsubseteq \tau$. We notice that $\mathrm{I} : \sigma \lesssim \mathrm{J} : \tau$ only if $\mathrm{I} \subseteq \mathrm{J}$. This apparently natural prerequisite has substantial consequences on the properties of $\lesssim$ because it ultimately enables width and depth extensions, which are not possible in the $\sqsubseteq$ preorder. For instance, we have $\{a\} : a \lesssim \{a, b\} : a + b$ whilst $a \not\sqsubseteq a + b$ (width extension). Similarly we have $\{a\} : a \lesssim \{a, b\} : a \,;\, b$ whilst $a \not\sqsubseteq a \,;\, b$ (depth extension). These extensions are desirable when searching for services, since every service offering more methods than required is a reasonable result of a query. The precise relationship between $\lesssim$ and $\sqsubseteq$ is expressed by the following statement.

Proposition 4.1. $\mathrm{I} : \sigma \approx \mathrm{J} : \tau$ *if and only if* $\sigma \asymp \tau$ *and* $\mathrm{I} = \mathrm{J}$.

The basic problem for querying Web service repositories is that, given a client $\mathrm{K} : \rho$, one wishes to find all the service contracts $\mathrm{I} : \sigma$ such that $\mathsf{actions}(\rho) \setminus \{\mathsf{e}\} \subseteq \bar{\mathrm{I}}$ and $\rho \dashv \sigma$. We attack this problem in two steps: first of all, we compute *one*

particular service contract $\overline{\mathrm{K}} \setminus \{\overline{\mathsf{e}}\} : \mathsf{D}^{\mathrm{K}}_{\rho}$ such that $\rho \dashv \mathsf{D}^{\mathrm{K}}_{\rho}$; second, we take all the services in the registry whose contract is larger than this one. In order to maximize the number of service contracts returned as answer to the query, the *dual* of a (client) contract $\mathrm{K} : \rho$ should be a contract $\overline{\mathrm{K}} \setminus \{\overline{\mathsf{e}}\} : \mathsf{D}^{\mathrm{K}}_{\rho}$ such that it is the smallest service contract that satisfies the client contract $\mathrm{K} : \rho$. We call such contract the *principal dual contract* of $\mathrm{K} : \rho$.

In defining the principal dual contract, it is convenient to restrict the definition to those client's behaviors ρ that never lead to $\mathsf{0}$ without emitting e. For example, the behavior $a\,;\mathsf{e} + b$ describes a client that succeeds if the service proposes $\overline{a}$, but that fails if the service proposes $\overline{b}$. As far as querying is concerned, such behavior is completely equivalent to $a\,;\mathsf{e}$. As another example, the degenerate client behavior $\mathsf{0}$ is such that no service will ever satisfy it. In general, if a client is unable to handle a particular action, like b in the first example, it should simply omit that action from its behavior. We say that a (client) contract $\mathrm{K} : \rho$ is *canonical* if, whenever $\rho \stackrel{\varphi}{\Longrightarrow} \rho'$ is maximal, then $\varphi = \varphi'\mathsf{e}$ and $\mathsf{e} \notin \mathsf{actions}(\varphi')$. For example $\{a, \mathsf{e}\} : a\,;\mathsf{e}$, $\{a\} : \mathsf{C}$, where $\mathsf{C} = a\,;\mathsf{C}$, and $\varnothing : \Omega$ are canonical; $\{a, b, \mathsf{e}\} : a\,;\mathsf{e} + b$ and $\{a\} : \mathsf{C}'$, where $\mathsf{C}' = a \oplus \mathsf{C}'$, are not canonical.

Observe that Lemma 2.1 also applies to contracts. Therefore it is possible to extend the notions in Definition 2.1, by replacing activities with contracts.

Definition 4.2 (Dual contract). *Let* $\mathrm{K} : \rho$ *be a canonical contract. The* dual *of* $\mathrm{K} : \rho$ *is* $\overline{\mathrm{K}} \setminus \{\overline{\mathsf{e}}\} : \mathsf{D}^{\mathrm{K}}_{\rho}$ *where* $\mathsf{D}^{\mathrm{K}}_{\rho}$ *is the contract name defined as follows:*

$$\mathsf{D}^{\mathrm{K}}_{\rho} \stackrel{\text{def}}{=} \begin{cases} \Omega & \text{if } \mathsf{init}(\rho) = \{\mathsf{e}\} \\ \sum_{\substack{\rho \Downarrow \mathrm{R} \\ \mathrm{R} \setminus \{\mathsf{e}\} \neq \varnothing}} \Big(\underbrace{\mathsf{0} \oplus}_{\text{if } \mathsf{e} \in \mathrm{R}} \bigoplus_{\alpha \in \mathrm{R} \setminus \{\mathsf{e}\}} \overline{\alpha}\,;\mathsf{D}^{\mathrm{K}}_{\rho(\alpha)} \Big) + \mathtt{OTH}_{\mathrm{K} \setminus \mathsf{init}(\rho)} & \text{otherwise} \end{cases}$$

$$\text{where } \mathtt{OTH}_{\mathrm{S}} \stackrel{\text{def}}{=} \underbrace{\mathsf{0} \oplus \bigoplus_{\alpha \in \mathrm{S}} \overline{\alpha}\,;\Omega}_{\text{if } \mathrm{S} \neq \varnothing}$$

Few comments about $\mathsf{D}^{\mathrm{K}}_{\rho}$, when $\mathsf{init}(\rho) \neq \{\mathsf{e}\}$, follow. In this case, the behavior ρ may autonomously transit to different states, each one offering a particular ready set. Thus the dual behavior leaves the choice to the client: this is the reason for the external choice in the second line. Once the state has been chosen, the client offers to the service a spectrum of possible actions: this is the reason for the internal choice underneath the sum $\sum$.

The contract $\mathtt{OTH}_{\mathrm{K} \setminus \mathsf{init}(\rho)}$ covers all the cases of actions that are allowed by the interface and that are not offered by the client. The point is that the dual operator must compute the principal (read, the smallest) service contract that satisfies the client, and the smallest convergent behavior with respect to a nonempty (finite) interface S is $\mathsf{0} \oplus \bigoplus_{\alpha \in \mathrm{S}} \overline{\alpha}\,;\Omega$. The $\mathsf{0}$ summand accounts for the possibility that none of the actions in $\mathrm{K} \setminus \mathsf{init}(\rho)$ is present. The external choice "+" distributes the proper dual contract over the internal choice of all the actions in $\mathrm{K} \setminus \mathsf{init}(\rho)$. For example, $\mathsf{D}^{\{a, \overline{a}, \mathsf{e}\}}_{a\,;\mathsf{e}} = \overline{a}\,;\Omega + (\mathsf{0} \oplus a\,;\Omega)$. The dual of a divergent

(canonical) client $\{\mathrm{A}\} : \mathsf{C}$, where $\mathsf{C} = a\,;\mathsf{e} \oplus \mathsf{C}$, is also well defined: $\mathsf{D}_{\mathsf{C}''}^{\{a,\mathsf{e}\}} = \overline{a}\,;\Omega$. We finally observe that the definition also accounts for duals of nonterminating clients, such as $\{\mathrm{A}\} : \mathsf{C}'$, where $\mathsf{C}' = a\,;\mathsf{C}'$. In this case, $\mathsf{D}_{\mathsf{C}'}^{\{a\}} = \overline{a}\,;\mathsf{D}_{\mathsf{C}'}^{\{a\}}$.

Similarly to the definition of contract names C_P, it is possible to prove that $\mathsf{D}_\rho^{\mathrm{K}}$ is well defined.

Lemma 4.1. *For every* $\mathrm{K} : \rho$*, the set* $\mathsf{cnames}(\mathsf{D}_\rho^{\mathrm{K}})$ *is finite.*

The property that $\overline{\mathrm{K}} \setminus \{\overline{\mathsf{e}}\} : \mathsf{D}_\rho^{\mathrm{K}}$ is the least dual contract of $\mathrm{K} : \rho$ follows.

Theorem 4.1. *Let* $\mathrm{K} : \rho$ *be a canonical contract. Then:*

1. $\rho \dashv \mathsf{D}_\rho^{\mathrm{K}}$*;*
2. *if* $\overline{\mathrm{K}} \setminus \{\overline{\mathsf{e}}\} \subseteq \mathrm{S}$ *and* $\rho \dashv \sigma$*, then* $\overline{\mathrm{K}} \setminus \{\overline{\mathsf{e}}\} : \mathsf{D}_\rho^{\mathrm{K}} \lesssim \mathrm{S} : \sigma$*.*

A final remark is about the computational complexity of the discovery algorithm. Determining $\lesssim$ is EXPTIME-complete in the size of the contracts [1], which has to be multiplied by the number of $\lesssim$-checks (to find a compliant service in the repository) to obtain the overall cost.

5 Conclusions

In this contribution we have studied a formal theory of Web service abstract (behavioral) definitions as normal forms of a natural semantics for BPEL activities. Our abstract definitions may be effectively used in any query-based system for service discovery because they support a notion of principal dual contract. This operation is currently done in an ad hoc fashion using search engines or similar technologies.

Several future research directions stem from this work. On the technical side, a limit of our technique is that BPEL activities are "static", *i.e.* they cannot create other services on the fly. This constraint implies the finiteness of models and, for this reason, it is possible to effectively associate an abstract description to activities. However, this impacts on scalability, in particular when services adapt to peaks of requests by creating additional services. It is well-known that such an additional feature makes models to be infinite states and requires an approximate inferential process to extract abstract descriptions from activities. Said otherwise, extending our technique to full CCS or π-calculus amounts to defining abstract finite models such that Theorem 3.1 does not hold anymore. For this reason, under- and over-estimations for services and clients, respectively, must be provided.

Another interesting technical issue concerns the extension of our study to other semantics for BPEL activities, such as the preorder in [6], or even to weak bisimulation (which has a polynomial computational cost). Perhaps one may use axiomatizations of these equivalences for determining the class of contracts. However it is not clear whether they admit a principal dual contract or not.

It is also interesting to prototyping our theory and experimenting it on some existing repository, such as `http://www.service-repository.com/`. To this aim we might use tools that have been already developed for the must testing, such as the concurrency workbench [11].

References

1. L. Aceto, A. Ingolfsdottir, and J. Srba. The algoritmics of bisimilarity. In D. Sangiorgi and J. Rutten, editors, *Advanced Topics in Bisimulation and Coinduction*, volume 52 of *Cambridge Tracts in Theoretical Computer Science*, chapter 3, pages 100–172. Cambridge University Press, 2011.
2. A. Alves et al. *Web Services Business Process Execution Language Version 2.0*, Jan. 2007. `http://docs.oasis-open.org/wsbpel/2.0/CS01/wsbpel-v2.0-CS01.html`.
3. A. Banerji, C. Bartolini, D. Beringer, V. Chopella, et al. *Web Services Conversation Language (*WSCL*) 1.0*, Mar. 2002. `http://www.w3.org/TR/2002/NOTE-wscl10-20020314`.
4. V. Benzaken, G. Castagna, and A. Frisch. CDuce: an XML-centric general-purpose language. *SIGPLAN Notices*, 38(9):51–63, 2003.
5. D. Beringer, H. Kuno, and M. Lemon. *Using* WSCL *in a* UDDI *Registry 1.0*, 2001. UDDI Working Draft Best Practices Document, `http://xml.coverpages.org/HP-UDDI-wscl-5-16-01.pdf`.
6. M. Bravetti and G. Zavattaro. A theory of contracts for strong service compliance. *Mathematical Structures in Computer Science*, 19:601–638, 5 2009.
7. N. Busi, M. Gabbrielli, and G. Zavattaro. On the expressive power of recursion, replication and iteration in process calculi. *Mathematical Structures in Computer Science*, 19(6):1191–1222, 2009.
8. G. Castagna, N. Gesbert, and L. Padovani. A Theory of Contracts for Web Services. *ACM Transactions on Programming Languages and Systems*, 31(5), 2009.
9. S. Chaki, S. K. Rajamani, and J. Rehof. Types as models: model checking message-passing programs. *SIGPLAN Not.*, 37(1):45–57, 2002.
10. E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. *Web Services Description Language (*WSDL*) 1.1*, 2001. `http://www.w3.org/TR/2001/NOTE-wsdl-20010315`.
11. R. Cleaveland, J. Parrow, and B. Steffen. The concurrency workbench: a semantics-based tool for the verification of concurrent systems. *ACM Trans. Program. Lang. Syst.*, 15(1):36–72, 1993.
12. R. De Nicola and M. Hennessy. Testing equivalences for processes. *Theor. Comput. Sci*, 34:83–133, 1984.
13. R. De Nicola and M. Hennessy. CCS without τ's. In *Proceedings of TAPSOFT'87/CAAP'87*, LNCS 249, pages 138–152. Springer, 1987.
14. S. Gay and M. Hole. Subtyping for session types in the π-calculus. *Acta Informatica*, 42(2-3):191–225, 2005.
15. M. Hennessy. *Algebraic Theory of Processes.* Foundation of Computing. MIT Press, 1988.
16. H. Hosoya and B. C. Pierce. XDuce: A statically typed XML processing language. *ACM Trans. Internet Techn.*, 3(2):117–148, 2003.
17. C. Laneve and L. Padovani. The *must* preorder revisited – an algebraic theory for web services contracts. In *CONCUR'07*, LNCS 4703, pages 212–225. Springer, 2007.
18. R. Milner. *A Calculus of Communicating Systems.* Springer, 1982.
19. H. R. Nielson and F. Nielson. Higher-order concurrent programs with finite communication topology (extended abstract). In *Proceedings of POPL'94*, pages 84–97. ACM Press, 1994.
20. S. Parastatidis and J. Webber. *MEP SSDL Protocol Framework*, Apr. 2005. `http://ssdl.org`.